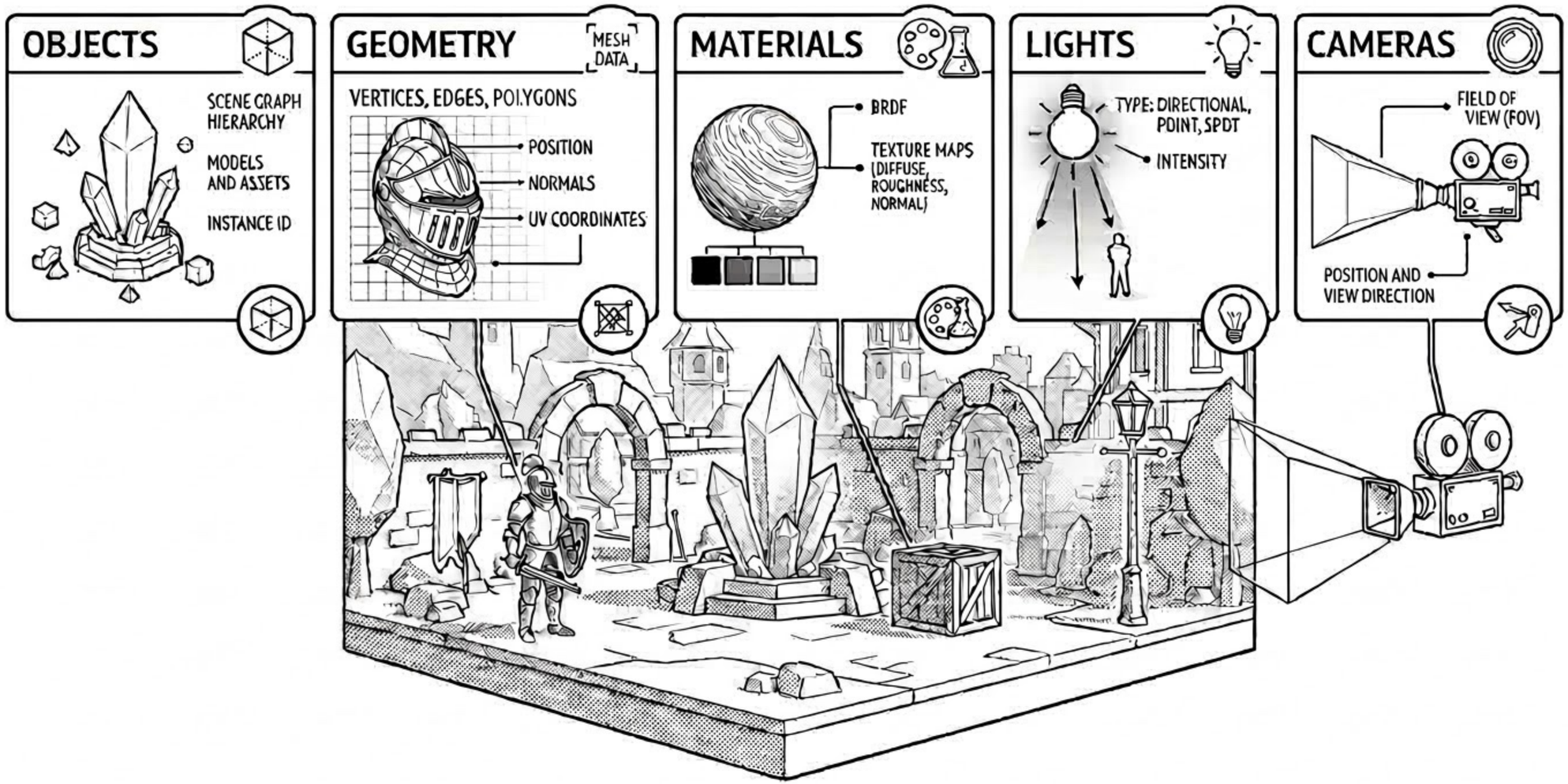
Lecture 18:

Neural Rendering and World Models

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

Representing scenes in CS248A

Your renderer took a scene representation as input, and generated an image that depicted the scene from the viewpoint of a virtual camera



How did we get this representations?

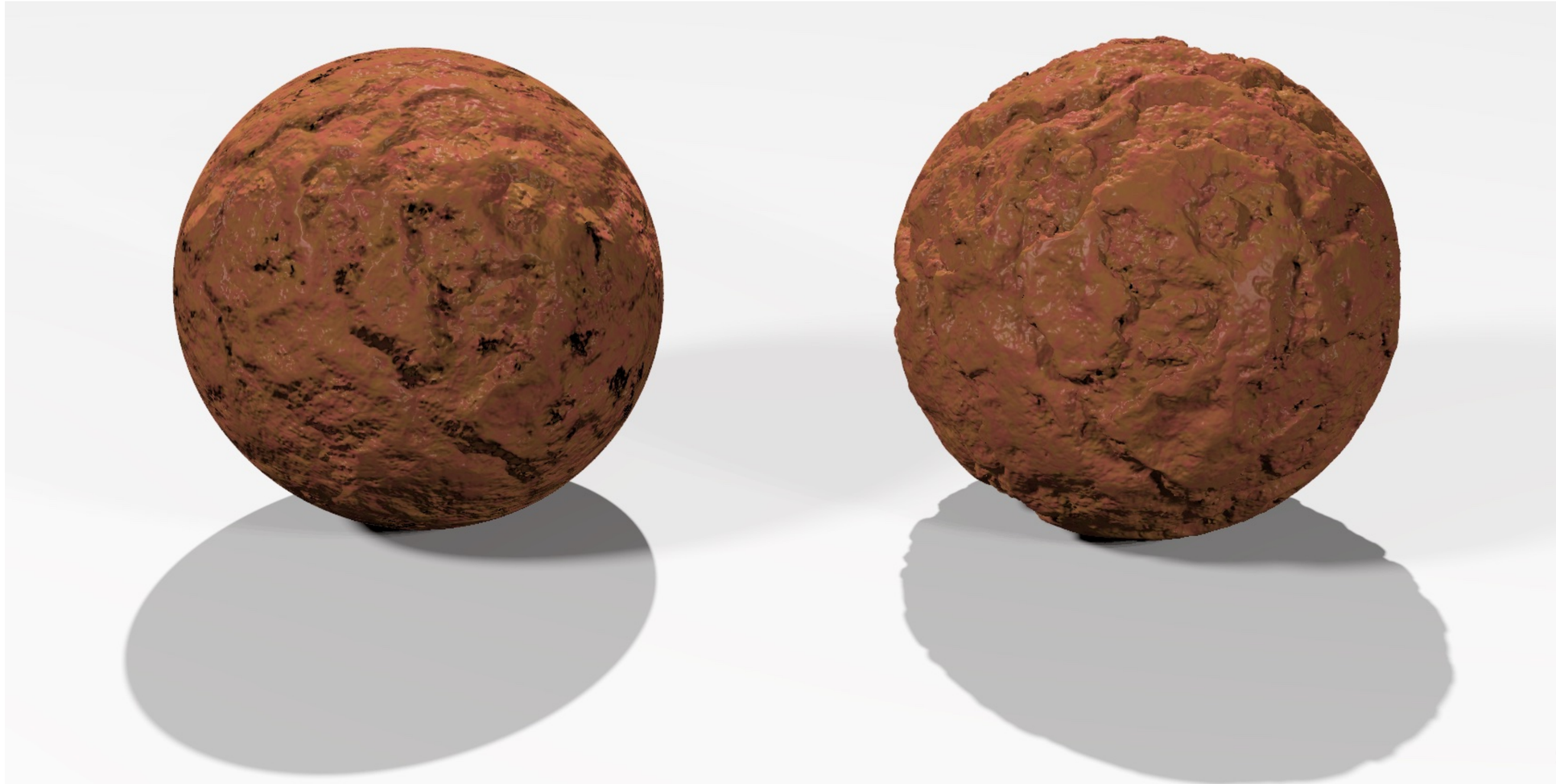
- We encoded our understanding of the world!

- We drew from:
  - Physical laws
  - Understanding of shape
  - Knowledge of how real world objects work (e.g., cameras, sensors)
  - Study of human perception

And from time-to-time we fell back to data

When it was tricky to model a phenomenon, or perhaps too expensive to model/simulate

Example: should we model the detailed geometry of a surface to encode rapidly varying surface normal… or should we use a low-poly surface approximation and "bake" normals into a texture (normal mapping)?



Trend: using a regular rendering process to produce an image (or "deep 2D frame buffer" of information about the scene at each screen sample) and then modifying the output of the renderer using a neural network

Back in 2009: morphological anti-aliasing (MLAA)

[Reshetov 09]

Aliased image
(one shading sample per pixel)

Zoomed views
(top: aliased, bottom: after MLAA)

After filtering using MLAA

By 2011…

- Entire course on different heuristics for hallucinating the missing content of under sampled images
  - Using different inputs produced by a renderer: low-res color, high-res albedo, motion vectors, recent frames, etc…

TALKS & COURSE NOTES

Download Extended Abstract [6.49 MB]

2:00	Introduction	Diego Gutierrez
2:05	A Directionally Adaptive Edge Anti-Aliasing Filter	Jason Yang
2:20	Morphological Anti-Aliasing (MLAA)	Alexander Reshetov
2:35	Jimenez's MLAA & SMAA (Subpixel Morphological Anti-Aliasing)	Jorge Jimenez
2:50	Hybrid CPU/GPU MLAA on the Xbox-360	Pete Demoreuille
3:05	MLAA on the PS3	Cedric Perthuis Tobias Berghoff
3:35	The Saboteur Anti-Aliasing (SPUAA)	Henry Yu
3:50	Break	
4:00	Subpixel Reconstruction Antialiasing (SRAA)	Morgan McGuire
4:15	FXAA 3.11 in 15 Slides	Timothy Lottes
4:30	Distance-to-edge Anti-Aliasing (DEAA)	Hugh Malan
4:45	Geometry Buffer Antialiasing (GBAA)	Emil Persson
4:55	Directionally Localized Anti-Aliasing (DLAA)	Dmitry Andreev
5:10	Anti-Aliasing Methods in CryENGINE 3	Tiago Sousa
5:25	Wrap-up and Discussion / Q & A	
5:30	Close	

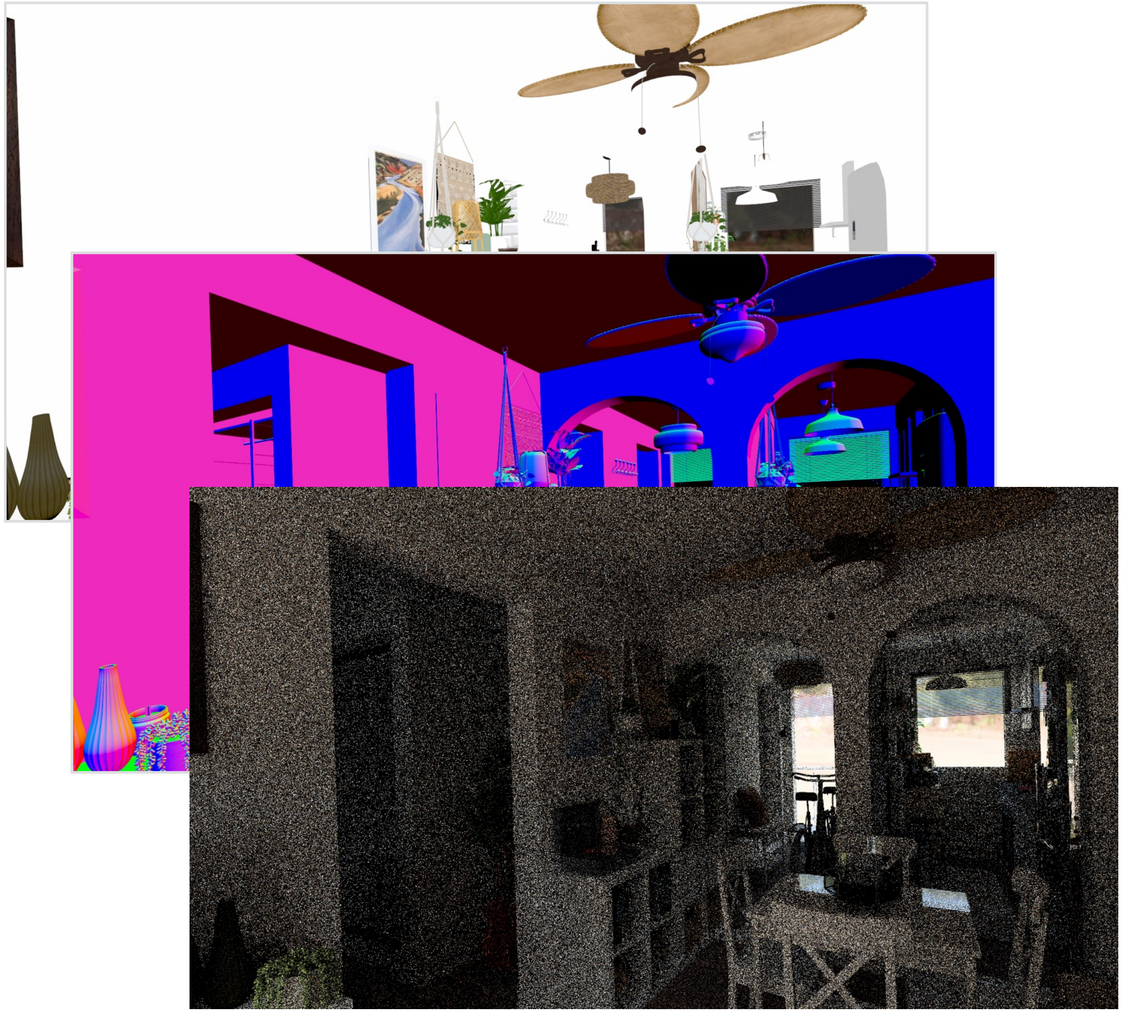
The 2026 version…

- Neural networks take renderer output and "beautify" the image
  - Spatial upsampling: increase its resolution
  - Temporal upsampling: increase frame rate
  - Anti-alias: remove aliasing due to undersampling
  - Denoising: remove noise due to variance in a ray tracer that does not have the budget to take enough samples to estimate radiance integrals

- Well known version: NVIDIA's "DLSS"
  - DLSS = Deep learning Supersampling… but now it does a lot more than supersampling, as noted above

Denoising example (from earlier lecture)

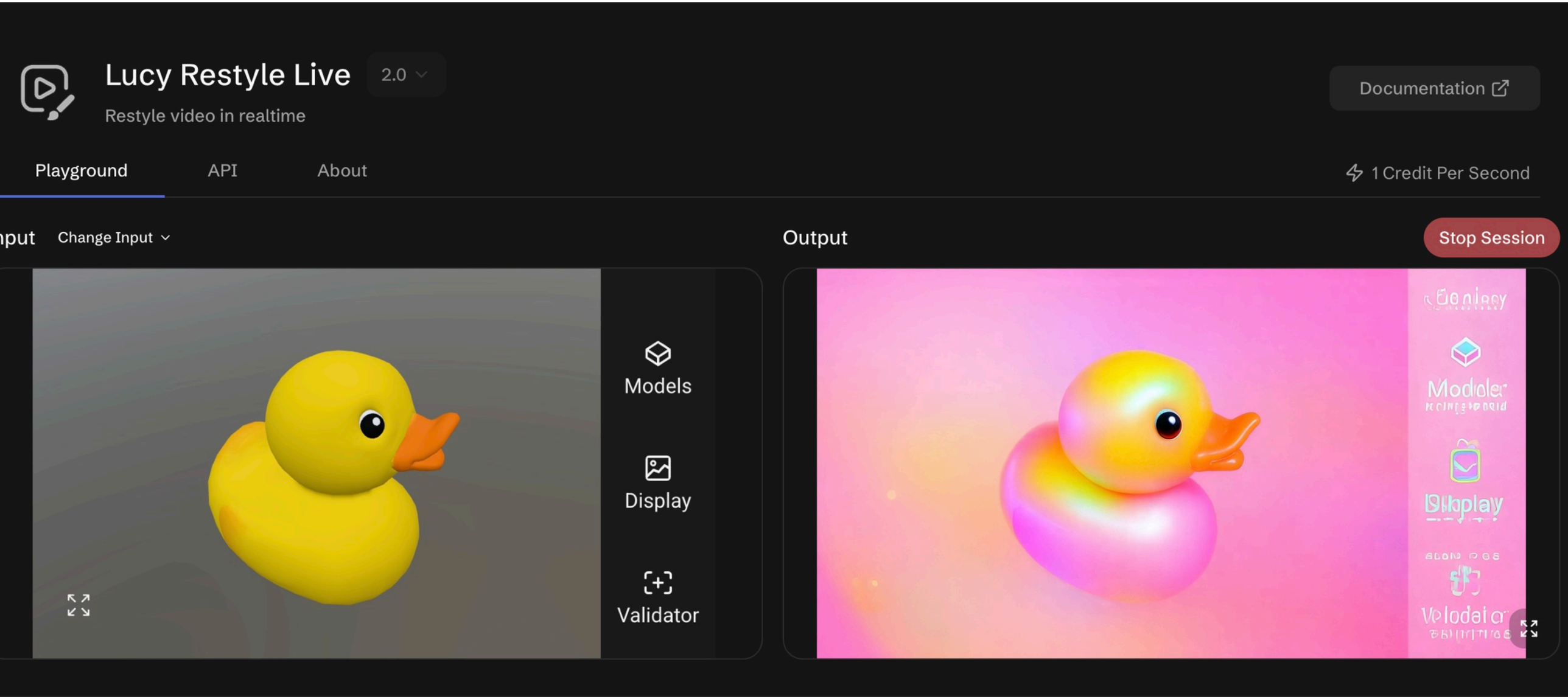
Inputs (produced by renderer cheaply)



Output: denoised image

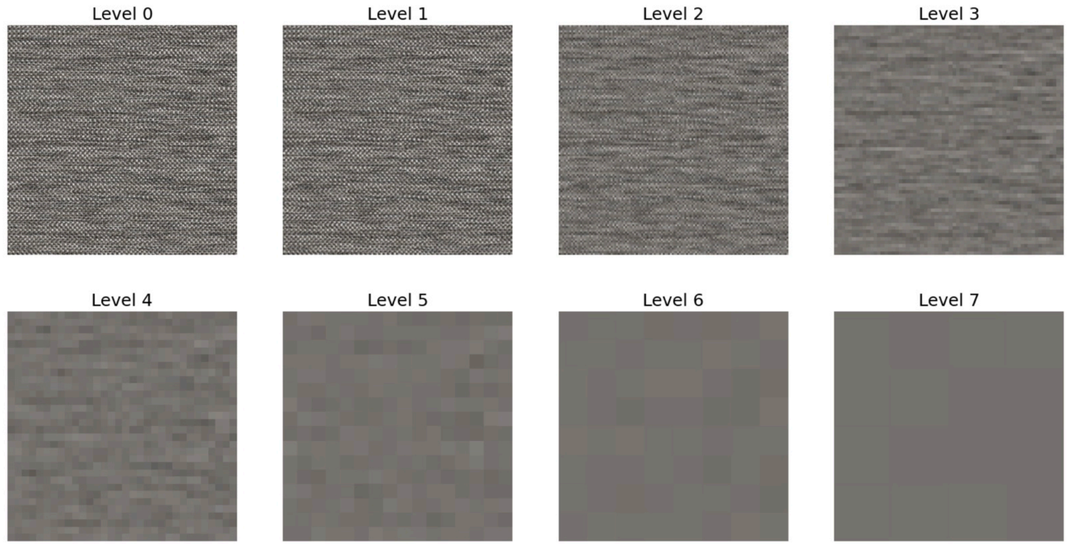
Why not just change the scene entirely?

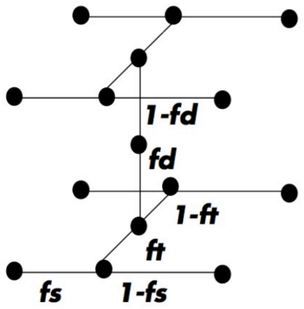
Fun with Decart.AI



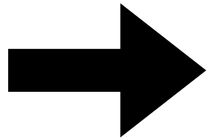
Trend: taking more sizable subroutines of the rendering process, and baking them into neural representations using optimization

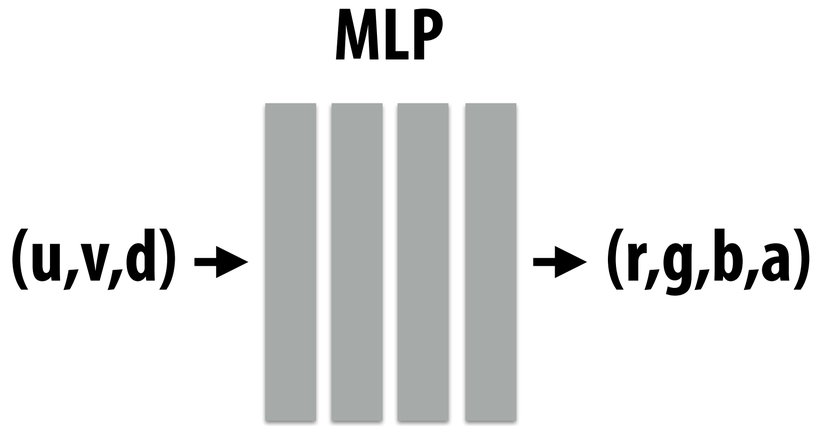
Example from assignment 2: distilling trilinear filtering of mipmap into a neural network





$$lerp(t, v_1, v_2) = v_1 + t(v_2 - v_1)$$



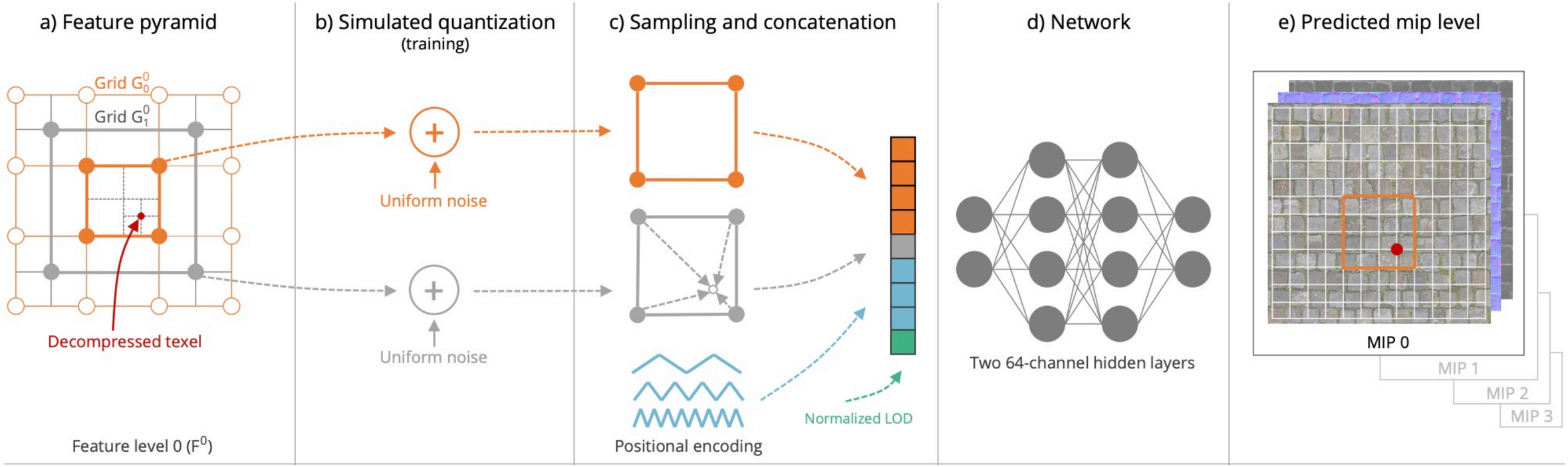


Why did we do this?

Neural texture compression: in practice

- **Store hierarchy of *features* (not RGBA texels): at low spatial resolution**
- **Decode features into traditional mip levels using trained neural network**
- **Do trilerp**

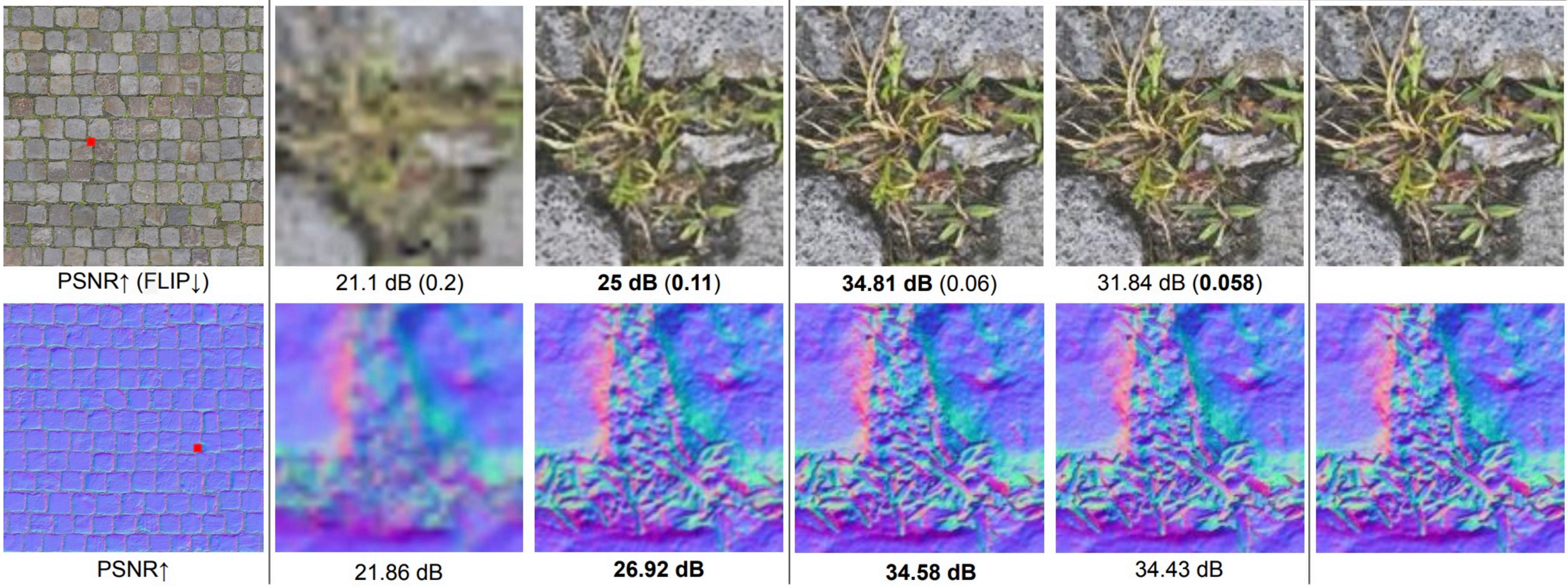
Feature level F^j	G_0^j grid resolution	G_1^j grid resolution	Predicted mip levels
0	256×256	128×128	0,1,2,3
1	64×64	32×32	4,5
2	16×16	8×8	6,7
3	4×4	2×2	8,9,10



Use case: compressing very high resolution textures

NTC = neural texture compression

BC = a traditional texture compressor "Block compressor"



Neural texture compression example

NTC. PSNR (↑): **22.0** dB, ꟻLIP (↓): **0.177**
4096 × 4096 at **3.8** MB.

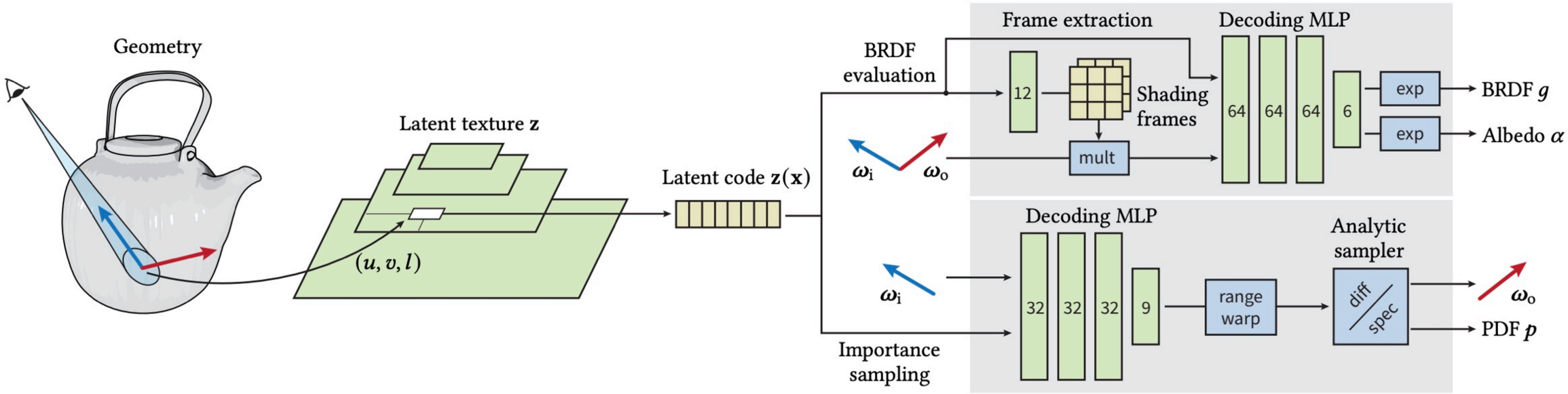
reference: not compressed
4096 × 4096 at 256 MB.

Neural texture: why?

- Reduce footprint: Compress complex textures

- But at increased computation cost!
  - Much more expensive to evaluate neural network than to perform trilinear or anisotropic filtering

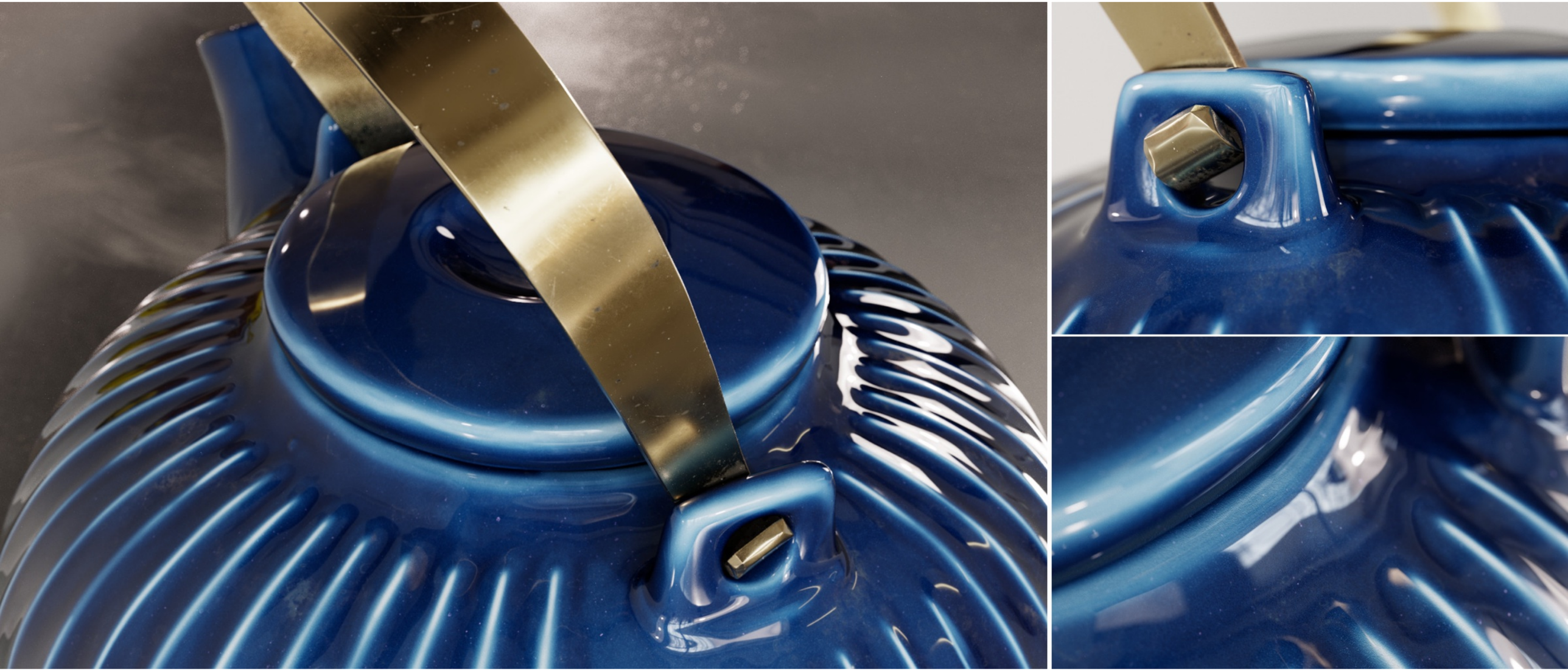
Neural materials



- Given hit point *x* and mip-level *l*
  - Retrieve features for that surface point and level-of-detail
  - Use neural network to decode features into BRDF value $f(\mathbf{x}, \omega_i, \omega_o)$
  - Note: BRDF *f*, is level-of-detail dependent, so can filter texture and the BRDF function itself
  - Optionally can choose indirect sampling direction w_o and PDF for choosing that direction

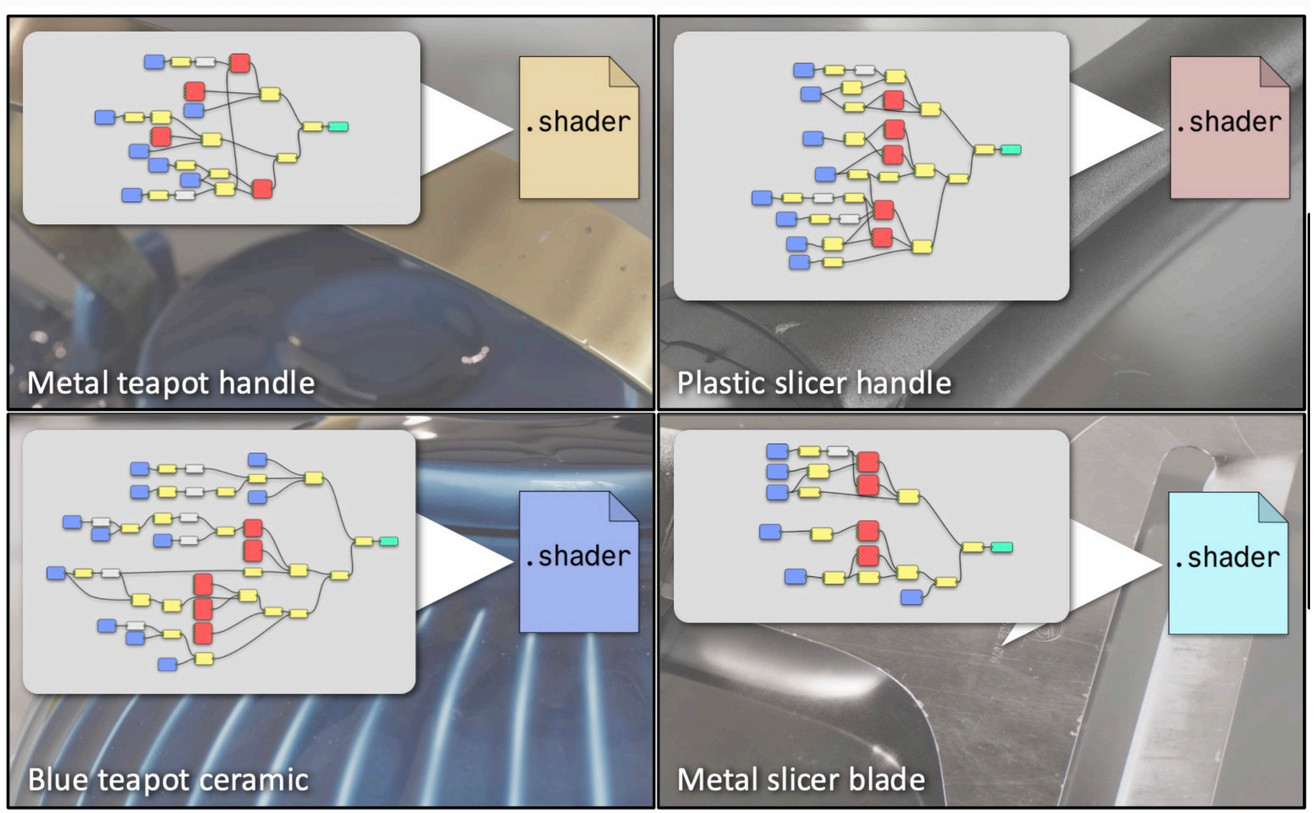
Neural materials

Move the entire BRDF evaluation into a neural network



Neural materials: why?

- Reduce footprint: Compress complex textures
  - Same as for neural texturing
- Reduce cost: Efficiently approximate complex shader programs
  - e.g., multi-layer materials
- Data-driven level-of-detail: learn a function that approximates anti-aliased results at many levels-of-level
  - Traditionally, graphics developers had to be very clever to come up with analytic approximations



Neural SDF representation

- Store neural features at every cell in an octtree
- Retrieve the features *f* given query point (x,y,z) Use a small neural network to decode *f* into distance(x,y,z)

- Why?
  - Same reasons… compression, level-of-detail
  - Suitability for reconstruction of 3D shape from measurements like photos

What about just replacing the whole rendering function?

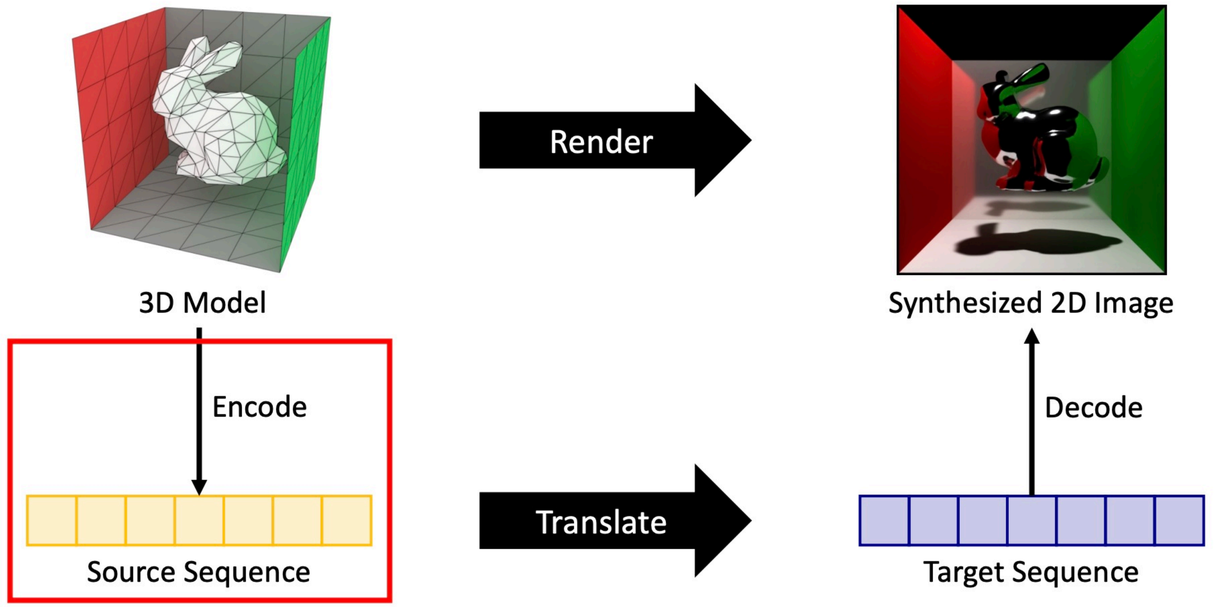
Recall: NeRF (neural radiance functions)

- Idea: bake L(x,w) for a scene into a neural network
- Query the neural network, instead of a recursive path tracer, when rendering an image
- Good for realistic synthesis of novel views, bad for many other things

How can we add more traditional forms of control over the scene?

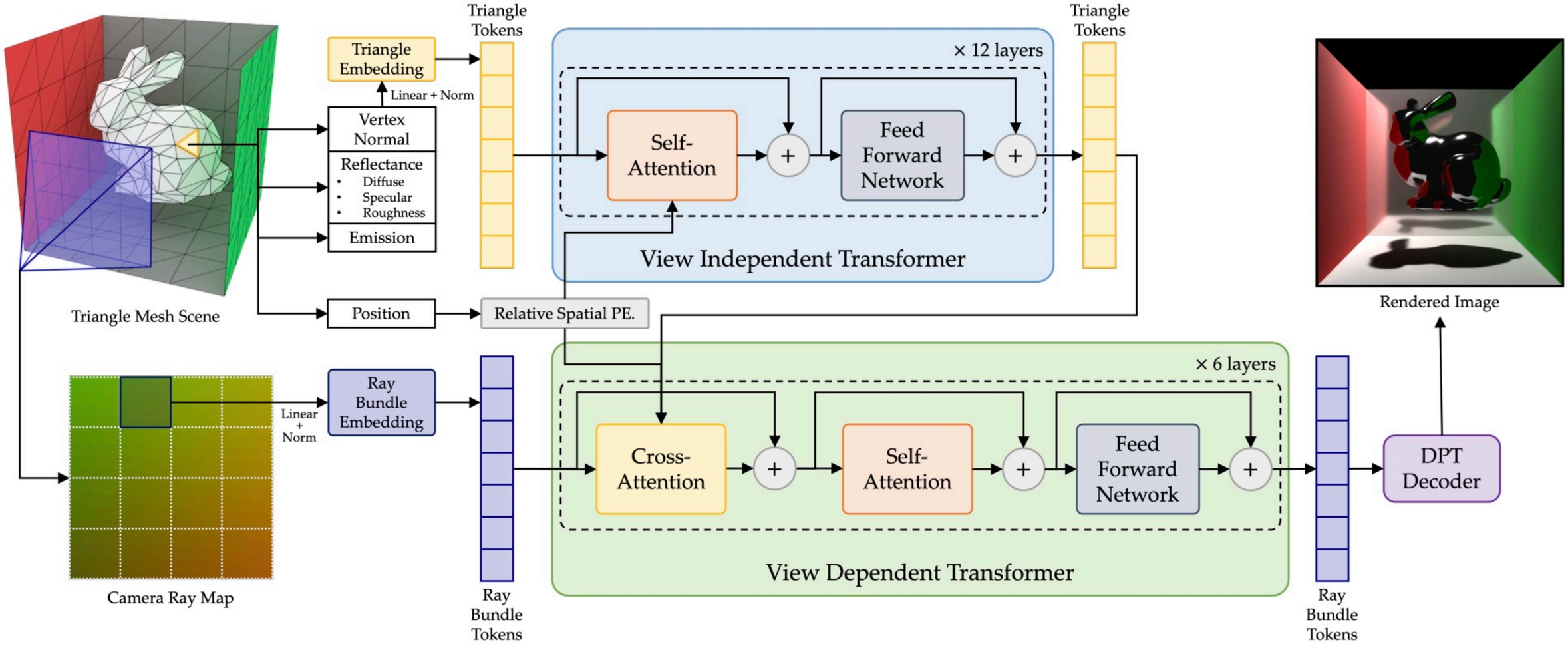
- **RenderFormer:**
  - **Input is a scene representation: list of triangles, material properties for triangles and info about camera**
  - **Output is an image**
  - **Inspiration: rendering is "translating a scene description to a 2D image"**

Idea: 3D Rendering = Translating 3D to 2D



RenderFormer: Translating scene primitives into pixels

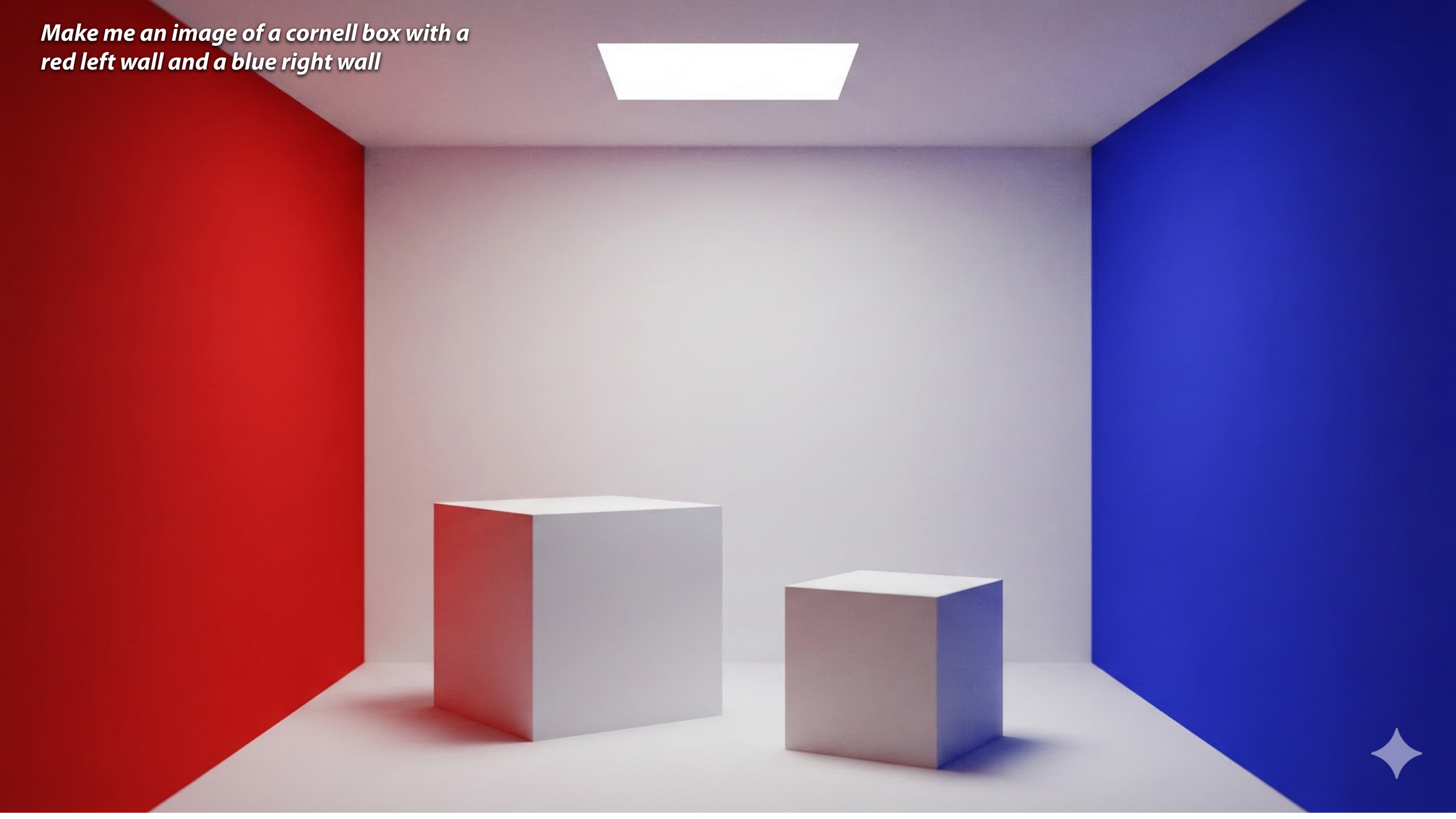
- Changing input scene parameters yields scene changes
- Pretrain model on synthetic data

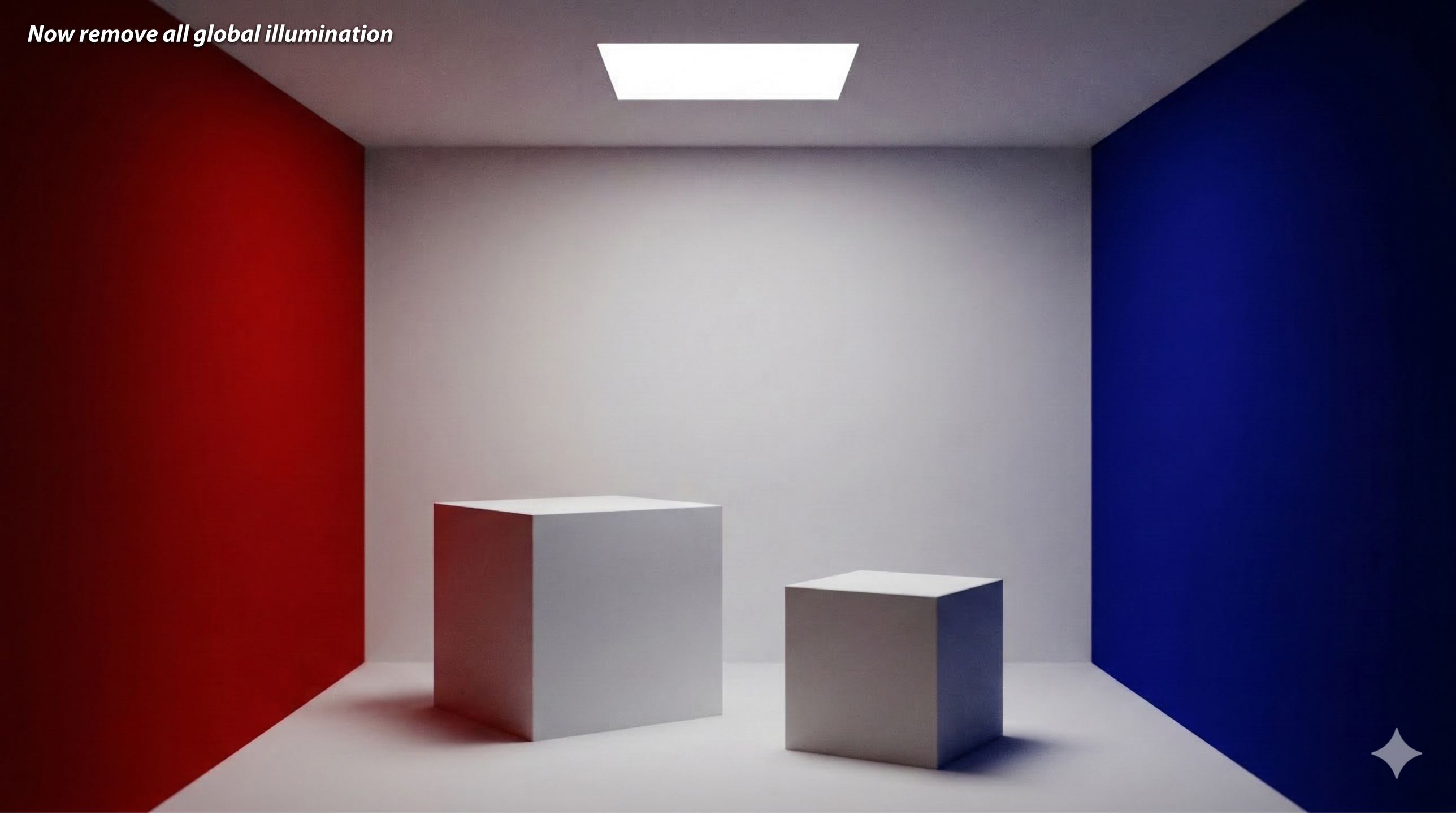


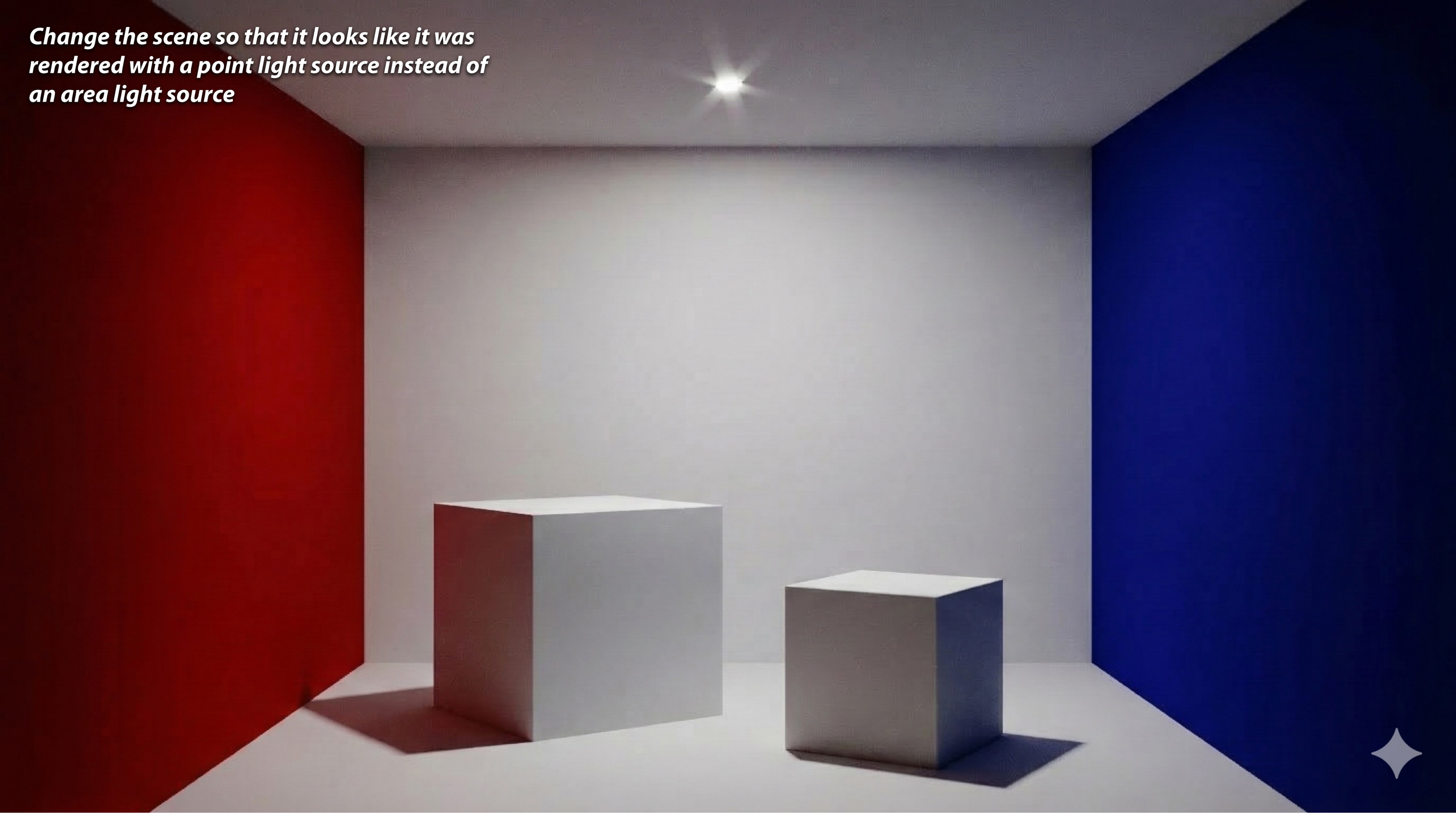
Text to image

please make an image of the Stanford CoDA building, with a view of the red stairwell on a day where it is raining large meatballs and spaghetti

Is Google's Nano Banana a renderer?


Make me an image of a cornell box with a red left wall and a blue right wall


Now remove all global illumination


Change the scene so that it looks like it was rendered with a point light source instead of an area light source

In a rendering with no global illumination and a single point light source located on the ceiling, should the ceiling get any illumination?

In a standard rendering with **no global illumination (GI)** and a single **point light source**, the ceiling should generally **not** receive any illumination.

Here is why, based on how light behaves in a simplified computer graphics model:

1. The Emission Pattern

A point light source radiates light in all directions (360 degrees) from a single infinitesimal point. However, in most 3D software and rendering engines, the light is calculated based on the "normal" (the direction a surface is facing) relative to the light's position. +1

2. The Surface Normal Problem

For a surface to be illuminated, it must have a line of sight to the light source, and its front face must be oriented toward that light.

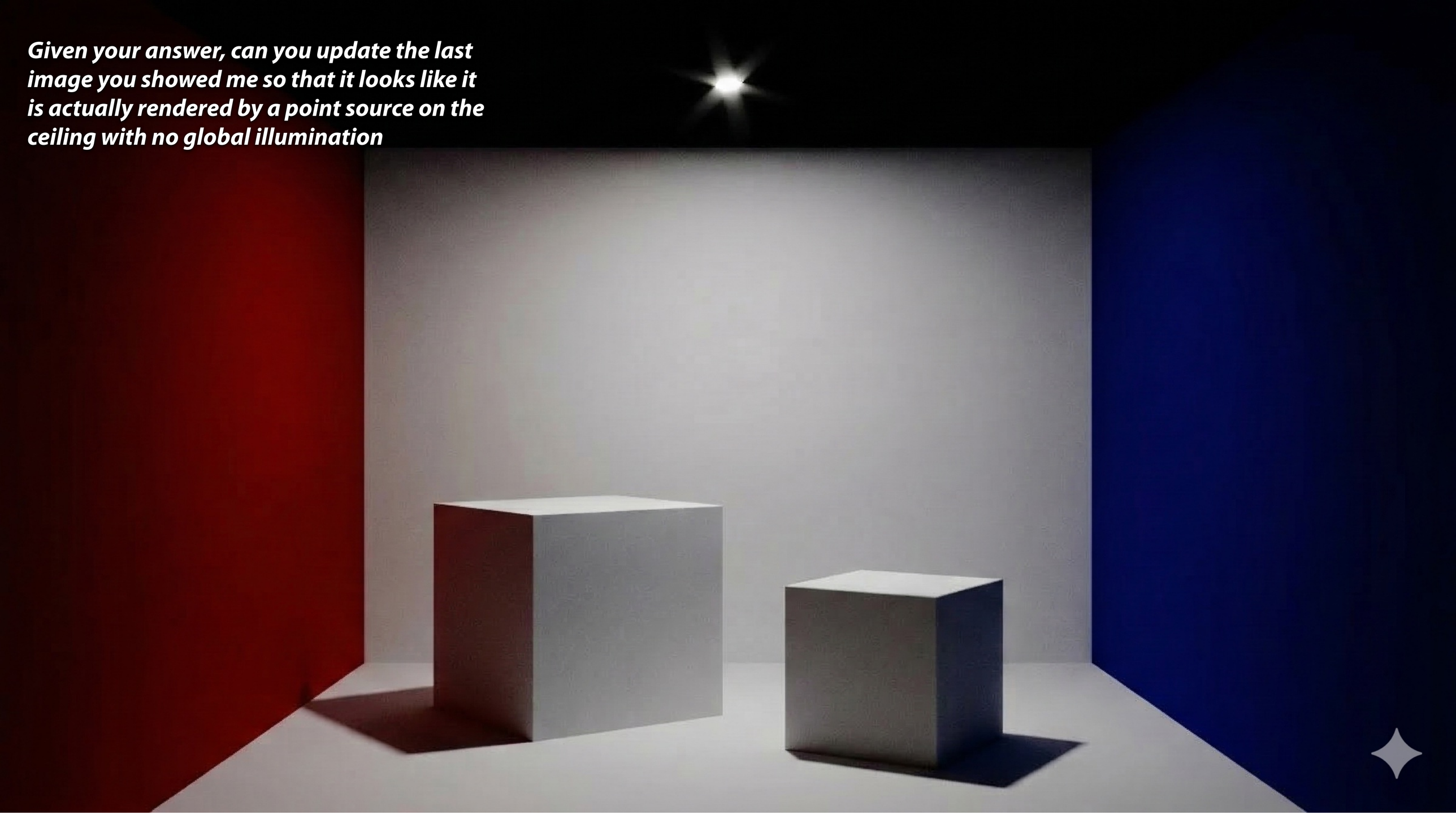
• • •

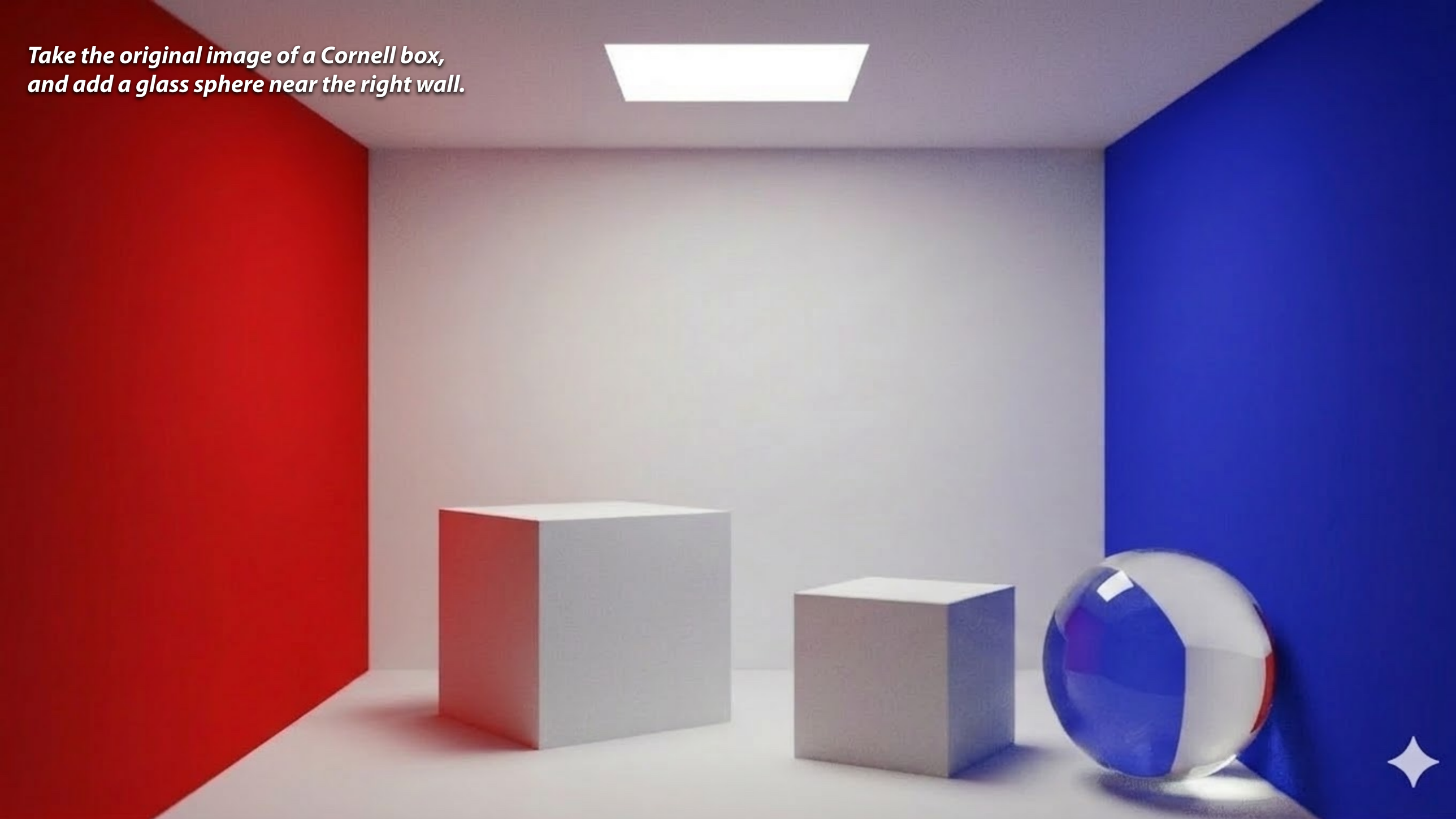
3. Lack of Light Bouncing

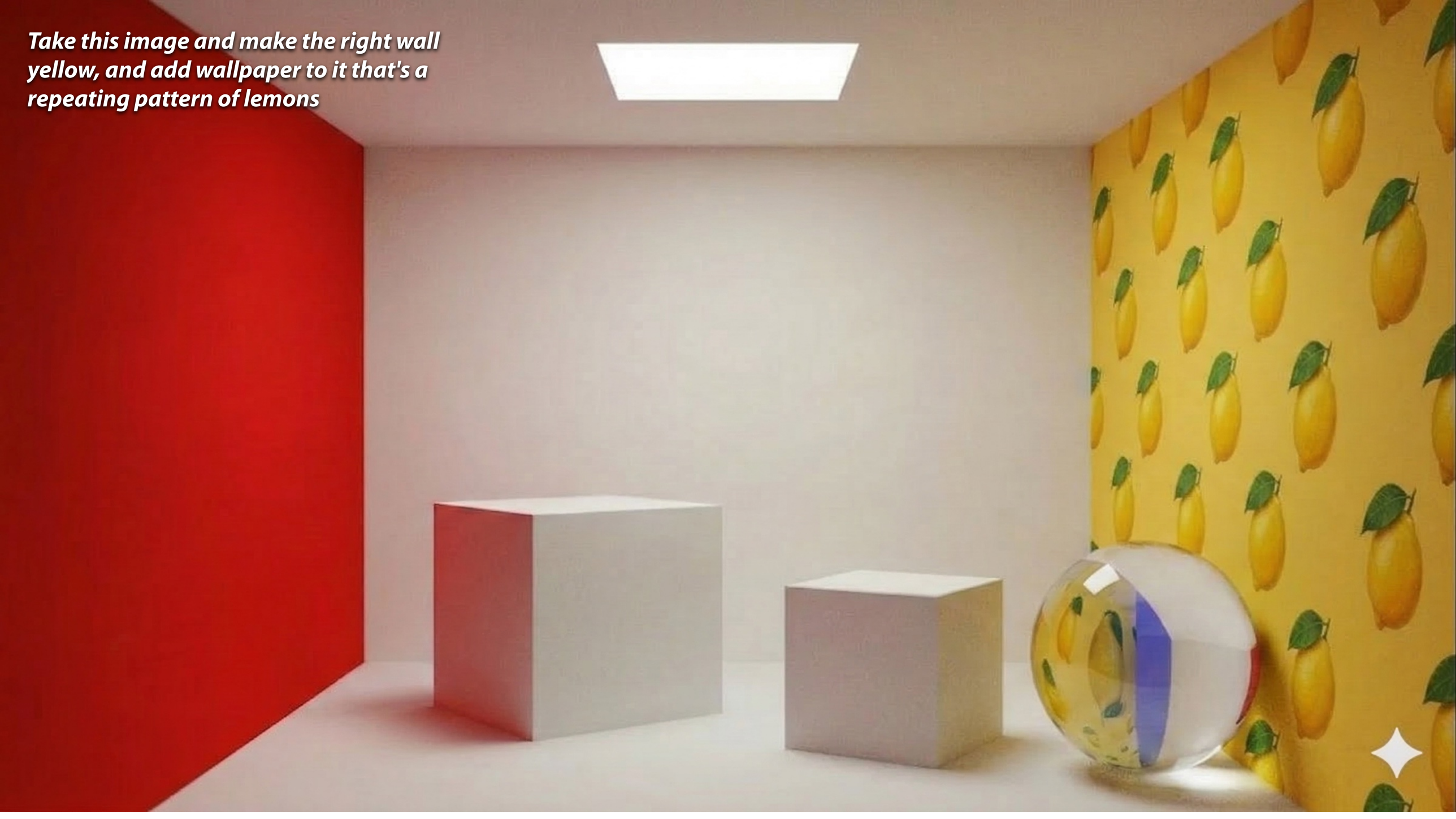
This is where the "no global illumination" part is critical. In the real world, the ceiling is bright because light hits the floor and walls and **bounces** back up.

- **With GI:** Light rays reflect off the white floor and red/blue walls, hitting the ceiling and making it visible.
- **Without GI:** Once a light ray hits the floor, it "dies." There are no secondary bounces to travel back up and illuminate the ceiling.

Given your answer, can you update the last image you showed me so that it looks like it is actually rendered by a point source on the ceiling with no global illumination




Take the original image of a Cornell box,
and add a glass sphere near the right wall.


Take this image and make the right wall yellow, and add wallpaper to it that's a repeating pattern of lemons


Add a monkey face object floating in the
middle of the scene and give it a shiny
metallic material (a perfect mirror material)


add a object that looks like Stanford professor Kayvon Fatahalian as a marble statue in the scene


Add a several students listening intently,
they should all be rendered with a material
that looks like a very fuzzy blue sweater,
almost like a hairball.


I mean college aged students.

And add the Stanford university tree logo in the middle of all the student's shirts.


Now render the scene from the perspective
of the lady on the left side of the image.

So is Nano Banana a renderer?

- Sure there are a few errors in physical correctness…
- But is it producing images depicting the given scene state?
  - What is the "scene definition" here?

Summary: many different takes on how to combine neural functions with the functionality of a renderer

- Post-process (enhance, stylize) output of a traditional renderer
- Replace/approximate subroutines of a rendering with neural function
- Replace/approximate the entire rendering function with a neural function
  - What control inputs does this neural function accept?
    - A more "traditional" scene representation
    - A text prompt or image?

- What are the pro's/cons of any particular configuration:
  - Rendering fidelity?
    - Note: includes ability to support hard fidelity problems like level-of-detail
  - Rendering cost?
  - Ability to control output?

One note on compute "cost"

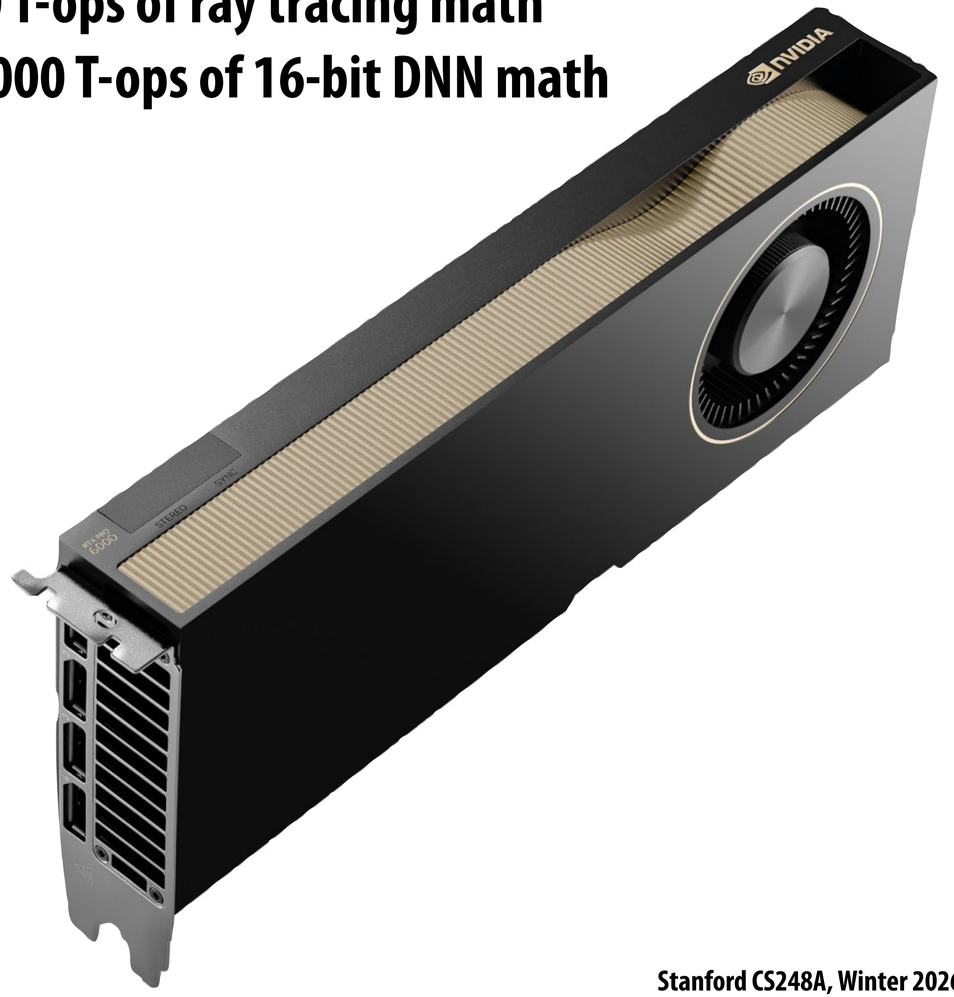
- A significant fraction of the compute capability of a modern GPU is in the special purpose acceleration of neural networks (Tensor Cores)

- So "runs on the GPU" is quickly becoming paramount to runs on the tensor cores

- The more of your algorithm you can offload to the tensor cores, the more you get to use this highly-efficient custom hardware
  - Overall wins come when the approximation via a neural approximation is not too much more inefficient than original algorithm

NVIDIA RTX 6000 Pro GPU

125 T-ops of FP32 math
380 T-ops of ray tracing math
~1000 T-ops of 16-bit DNN math



One note on authoring "cost"

- At this point in the course you have all the basics needed to generate physically accurate, beautiful images

- The field knows how to render beautiful pictures. But consider what is the barrier to making your path tracing renderer generate this scene

- Traditional representations that are controllable, interpretable, and editable are laborious to create

- Generative AI techniques for synthesizing 3D geometry, textures, materials, etc will make this easier

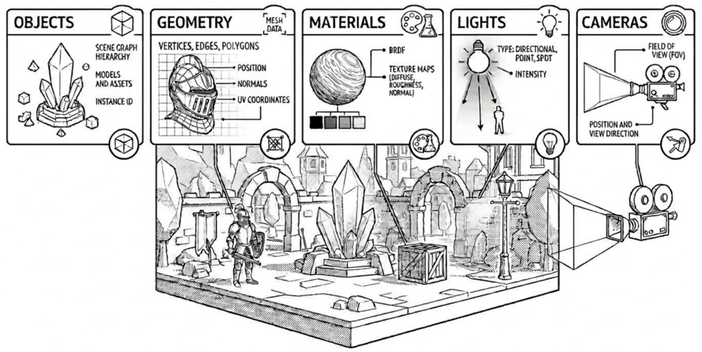
- But why bother at all with these "intermediates"? (class discussion)

What about replacing the game engine entirely?

User input

Physics system
AI system for bots
Game logic scripts

3D scene
(Surfaces, materials, lights, camera)

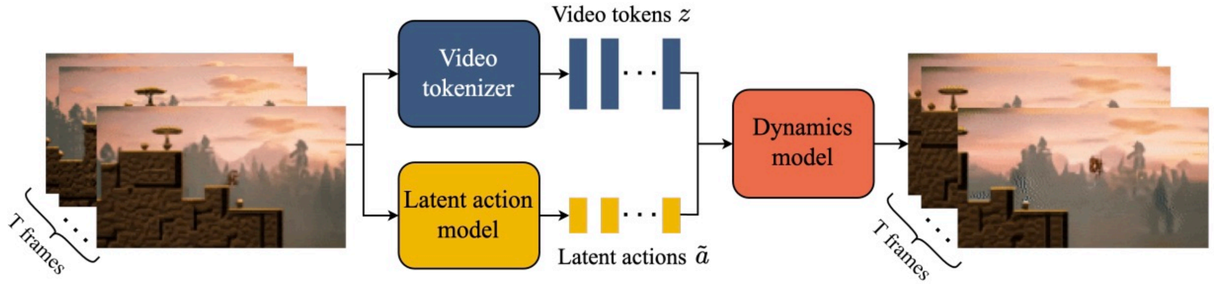


Renderer

2D image

"World model" = predicting time evolution of 3D environment in a neural network
(And these days the renderer gets rolled in as well)

Examples: Deepmind's Genie / NVIDIA COSMOS



Input: scene prompt + action at each frame

Output: next frame RGB
(and potentially additional per-frame data: embeddings, attributes)

Basic idea

- Given prompt (global description of scene)
- And outputs of the world model so far
  - Last N frames?
    - RGB? Neural features, etc.
  - A select number of "keyframes"
  - 3D reconstructed world from last N frames?
- And the action from the user

- Predict the next frame

- Key question: how to represent world state? (see web demos…)

Takeaways

- Huge current interest in world models
  - For interactive entertainment
  - For simulation for training robots

- Trained on large amounts of video data
  - Sometimes annotated with actions (e.g., video games, or robot manipulating worlds)

- We'll discuss opinions/takes in class: no slides ;-)